Ontology based Comprehensive Architecture for Service Discovery in Emergency Cloud

J.Uma Maheswari[#1], Dr.G.R.Karpagam[*2]

[#]Assistant Professor (Senior Grade), CSE Department, PSG College of Technology, Coimbatore-641004, Tamilnadu, India.
[1]uma_ftt@yahoo.co.in

[*]Professor, CSE Department, PSG College of Technology, Coimbatore-641004, Tamilnadu, India.
[2]grkarpagam@gmail.com

Abstract — Disasters such as Tsunami, Landslides, and cyclones occur frequently and a strong emergency management system is required to manage such situations. These kinds of crisis circumstances are expected to increase in future. The role of Information and communication technology can largely aid in handling calamities and provide first aid support. The characteristics of Cloud computing such as sharing on demand, connecting communities and offering everything as a service clearly indicate that it can contribute to crisis without affecting business continuity. Hence efforts have taken to articulate web services and the cloud infrastructure as ontology, in the perspective of emergency management which can improve the understanding of this proposed agent based comprehensive architecture.

Keyword- Cloud Computing, Web Services, Semantics, Service Discovery, agents

I. INTRODUCTION

Emergency management is a discipline which is used to manage the hazardous situations from food provisioning to medical treatment by applying science and technology. Emergency management has gained importance due to frequent situations such as earthquake, floods, hurricanes, volcanic eruptions and wild fires all over the globe. These situations expect efficient communication and rapid transportation which can be satisfied by group communication that involves entities like people, organization, events, locations and essential services. A recent development of i-phones and androids increases the social interaction by using social networks [1]. The technology behind the social network is cloud computing. Cloud computing [2] is Internet based development and use of computer technology whereby dynamically scalable and often virtualized resources are provided as a service over the Internet. Consumers of Cloud computing will not compute their own computer, but move their programs and data to the Clouds consisting of computation and storage utilities provided by third parties. Cloud computing providers publish Cloud services over the Internet, and consumers normally access these services provided by Cloud application layer through web-portals.

Another revolution in World Wide Web is Semantic Web [3] which has gained the attention of lot researchers. Semantic Web is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites. Ontologies are considered one of the pillars of the Semantic Web. Ontologies include computer usable definitions of basic concepts in the domain and relationship among them. The availability of machine-readable metadata would enable automated agents and other software to access the web more intelligently. The agents would be able to perform tasks and locate related information automatically on behalf of the user. Semantic search [4] locates information automatically by using the two types of ontologies namely domain ontology and service ontology. Domain Ontology is a conceptualization of a specific domain which enables the users to recognize the importance and relation between the terms and concepts of that domain. Service ontology is a conceptualization of set of services related to that particular domain. A third type of ontology, Cloud Ontology, a conceptualization of platforms, infrastructure and software that are supported and available in the cloud is included in this paper.

To date, however, there is no discovery mechanism for searching different kinds of Clouds. Cloud consumers generally have to search for appropriate Cloud services manually. Even though there are many existing generic search engines that consumers can use for finding Cloud services, these engines may return URLs containing not relevant web-pages to meet the original service requirements of consumers. Intuitively, visiting all the web-page can be time-consuming job. Whereas generic search engines (e.g., Google, MSN, etc) are very effective tools for searching URLs for generic user queries, they are not designed to reason about the relations among the different types of Cloud services and determining which services would be the best or most appropriate service for meeting consumer‘s service requirements. Hence, service discovery mechanisms for reasoning about similarity relations among Cloud services are needed.

Web service [5] is a programmable Web application that is universally accessible through standard Internet protocols, such as Simple Object Access Protocol (SOAP). Web service technology is becoming more and more popular in many practical application domains, such as electronic commerce, flow management, application integration, etc. It presents a promising solution for solving platform interoperability problems encountered by the application system integrators. With the ever increasing number of functional similar web services being made available on the Internet, how to distinguish the best Web service from others becomes an urgent problem to be solved. Web Service Selection is a key component in service-oriented architecture. The selection of web service is usually based on the functional requirements of the consumer but those web services may not able to provide the quality the consumer expectations. Consumer requirements may include not only functional aspects. It considers Non-functional demands, in addition to functional capabilities to provide a good quality of service to the consumer.

The Web Service Selection based on Non-functional parameters becomes the challenging task in current trend. Quality of Service [13] is an aggregated metric for describing characteristics of systems in areas, such as networks and distributed systems. A new approach has been proposed to discover the suitable web services based on functional and non functional parameters in a cloud environment for Emergency Management.

II. LITERATURE SURVEY

Arkaitz Ruiz-Alvarez et al [12] proposes the practices for Automated Approach to Cloud Storage Service Selection which focuses on Cloud services identification, publication, Cloud storage and network service (IaaS level),but its lacks the focus on ontologies. Miranda Zhang, Rajiv Ranjan, Armin Haller et al [7] Provides the OWL based ontology called cloud computing Ontology (CoCoOn) that defines functional and non functional concepts, attributes and relations of infrastructure services. The details about the Paas and Saas are not included in the ontology. Jaeyong Kang, Kwang Mong Sim [8] presents a four-stage, agent-based Cloud service discovery protocol. Utilizing an ontology description, in which each resource is described semantically and relatively to other resources. They developed a multi-agent system that uses ontology-based matching. The matching is based on input parameter value. The non functional parameters have not been taken for consideration. G. Vadivelou, E. IIavarasan, R. Manoharan, P. Praveen proposes a new architecture[9] called the Delegation Web Service (DWS) for selecting the web service with maximum load balancing. The load balancing is achieved by grouping the web services of similar type from the registry by the DWS for each consumer‘s request.

Alireza Zohali, Dr.Kamran Zamanifar provides the match making model[11] which considers the functional as well as the non functional parameters. The service matching method of this paper composed of two phases. In the first phase, the proper web services that satisfy the functionality matching of the desired service is found, and in the second phase, the best one is selected from Web Services set obtained in the first phase using semantic filtering. In the article [15], the analysis of requirements for a broker that performs discovery and mediation between agents and Web services are explained.

Octavio Gutierrez-Garcia and Kwang-MongSim say that, Selforganizing systems are composed of interacting individuals (e.g., agents) [17,18]. The interaction among individuals adapts and evolves the system to achieve an objective (e.g., Cloud service selection). The constitution and objective of the system emerge from the interaction of its members [16]. The emergent constitution is determined by the feedback obtained through the free interaction of nearby members. Thus, each member determines the partners to be linked to, by using local rules. Events in the system i.e., A Cloud consumer submitting requirements, produce changes that are initially managed by directly connected members, which replicate the new constitution of the system to indirectly connected members. This paper overcomes the limitations of the above related works and proposes the new comprehensive architecture for web service discovery.

UmeshBellur, RoshanKulkarni proposes techniques for semantic description and matchmaking of services [10]. These techniques use semantic concepts from Ontologies [19] to describe the Inputs, Outputs, Pre-conditions and Effects (IOPE) of a service. The discovery process involves the matchmaking of the semantic descriptions offered by the client and the provider. The algorithm takes an OWL-S [20] *Query* from the client as input and iterates over every OWL-S *Advertisement* in its repository in order to determine a match. An *Advertisement* and a *Query* match if their Outputs and Inputs, both, match. The algorithm returns a set of matching advertisements sorted according to the degree of match such as Exact, Plugin, Subsume, and Fail. These four degrees as ranked as: *Exact >Plugin >Subsumes >Fail*.

III. COMPREHENSIVE ARCHITECTURE

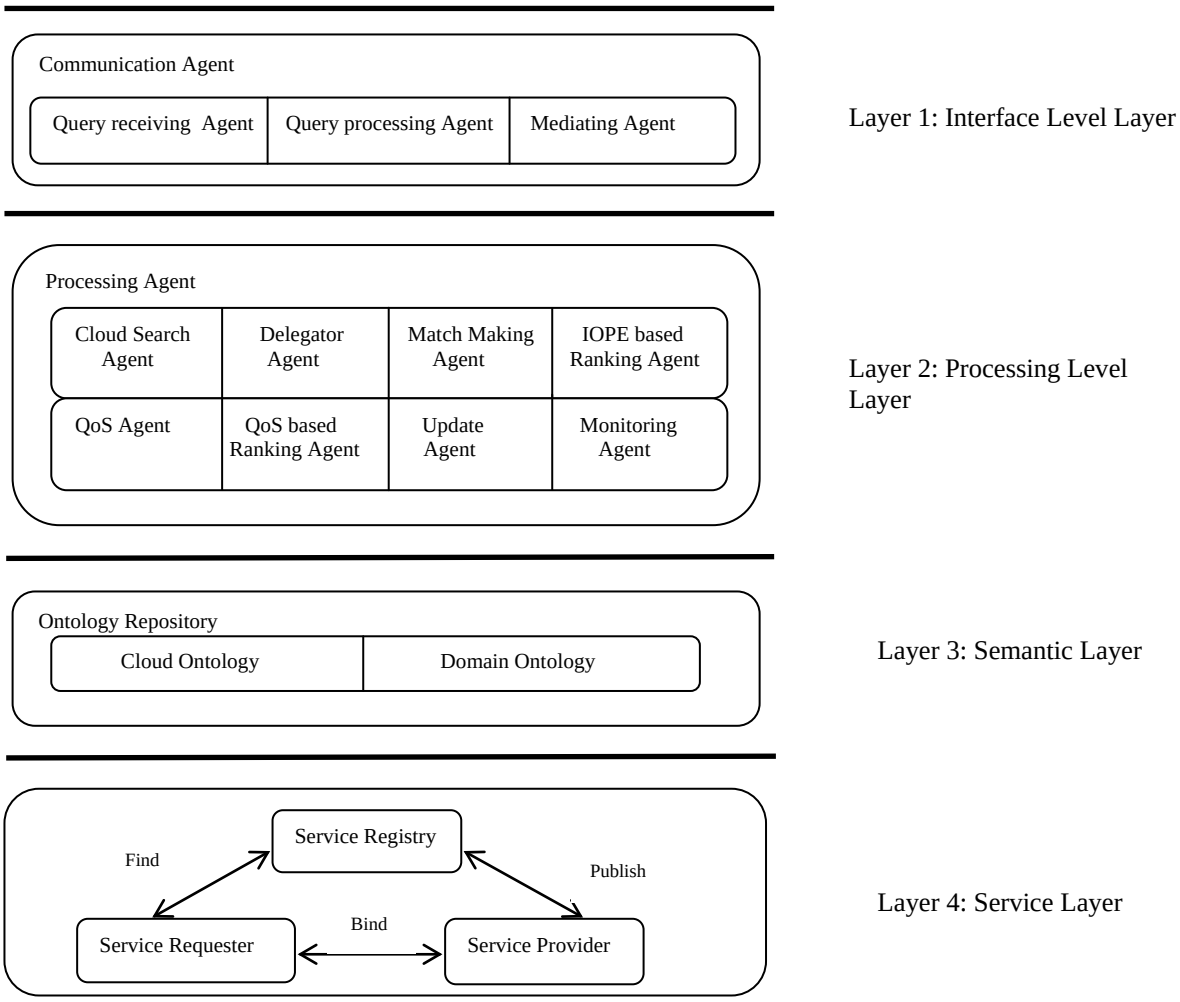

Fig. 1. Comprehensive Architecture

The objective of this paper is to provide ontology based Emergency Management in cloud. Agent based architecture is proposed to achieve the goal. A multilayered comprehensive architecture is shown in Fig.1 and each layer has its own responsibilities. Layer 1 emphasizes on user query and the processing of user query. Layer 2 constitutes several processing agents and it is responsible for emergency service discovery in cloud. Layer 3 explores the cloud and domain ontologies that are needed to provide the information about the emergency cloud and their domains. The Last Layer 4 comprises the actual services which are needed to be discovered.

A. Interface Level Layer

The topmost Layer Interface level Layer emphasizes on user Query and the processing of such queries. It comprises of three sub agents namely (i) Query receiving agent (ii) Query Processing Agent (iii) Mediating agent. Query Receiving Agent receives the bulk request from the user. The query undergoes for various pre processing steps like stemming, stop word removal and splitting. The pre-processed query is sent to the service history. If the query is matched with the query in the history then the corresponding services are retrieved. Otherwise the pre-processed query is sent to the Search Agent.

Receiving agent gets the query from the user and the query is stemmed by using the reduced version of porter stemmer algorithm. The Porter Stemmer [14] is a conflation Stemmer developed by Martin Porter at the University of Cambridge in 1980. The Stemmer is based on the idea that the suffixes in the English language (approximately 1200) are mostly made up of a combination of smaller and simpler suffixes. The Porter Stemmer is a very widely used and available Stemmer, and is used in many applications. The pseudo code for the stemmer is given in Fig.2.

```
If (suffix of a word==”sses”) replace it by “ss”. (caresses -> caress)
Else If (suffix of a word==”ies”) replace it by ”i”.( ties -> tie)
Else If (suffix of a word==”s”) remove. (cats -> cat)
Else if (suffix of a word==”ed”) remove.( plastered-> plaster)
Else if (suffix of a word==”ness”) remove.( goodness -> good)
Else if (suffix of a word==”ing”) remove.(singing->sing)
Else if (suffix of a word==”ful”) remove.(hopeful -> hope)
Else if (suffix of a word==”icate”) replace it by “ic”.(triplicate -> triplic)
Else if (suffix of a word==”alize”) replace it by “al”.(formalize -> formal)
Else if (suffix of a word==”ation”) replace it by “ate”.( predication -> predicate)
```

Fig.2. Stemming Procedure

After stemming the query is given to the next agent ie Query Processing agent. QPA does one of the pre processing steps ie stop word removal. The procedure for this is given in Fig.3(a).The lists of stop words are initialized and word that is extracted from query is matched against this list. If there is any match then the word is removed from the query. After stop word removal mediating agent splits the user bulk query using the delimiter ‘,’. Fig.3 (b) explains the overall processing.

```
stop_word[ ][10]={"i", "a", "about","an","are","as","at","be",
"by","com","for","from","how","in", "is","it","of","on",
"or",  "that","the","this","to","was","what","when",
"where","who",  "will","with"}
  for all word_in in a query
  for all word_out in a stop_word list
  if(word_in of a query == stop word_out)
   remove the word
```

Fig.3(a) Stop word removal

```
For all individual query in bulk query

If(match(user query, query in  service history)
{
   find the cloud name;

    retrieve the service;

}

Else

      Transfer the request to Search Agent
```

Fig.3(b) Query Processing

The Service history has three fields namely (i) query (ii) cloud name (iii) service retrieved. Each individual query form bulk query is matched with the query in the service history. If there is a match the service is retrieved, else the query is transferred to search agent of the next layer. The example is given in Table I.

TABLE I
Query Processing Agent

Invoke **QPA**(User_Query)	Invoke QPA(need doctors, volunteers, food for people)
Generate **call_id**	236721
Invoke **stemming**(User_Query)	R1 = need doctor, volunteer, food for people
Invoke **stop_word_removal(**R1)	R2= need doctor, volunteer, food people
Invoke **query_split**(R2)	R31 = need doctor R32 = volunteer R33 = food people
Invoke **service_history**(R31,R32,R33)	No Match
Invoke **search_agent**(R31,R32,R33)	Search agent is invoked with splitted query

B. Processing Level Layer

1) Search Agent (SA): The search agent retrieves the name of the clouds for each query in which the requested services resides. The splitted query from Layer 1 is passed to Search Agent of the Processing Layer. The procedure is shown in Table II. According to the number of queries the algorithm is repeated. , Therefore algorithm is repeated thrice because the numbers of queries are three. For a single query all the clouds are searched to find the matching of query with services registered under the cloud if there is a match then the cloud name is returned to the Delegator Agent. In the example, for the query volunteer the matched clouds are cloud1, cloud2, cloud5.

TABLE II
Search Agent

Invoke **search_agent**(Splitted Query)	Invoke search_agent(R31,R32,R33)
Invoke **search_cloudontology**(individual Query) return **cloud_names**	Invoke search_cloudontology(R32) return cloud1,cloud2,cloud5
Invoke **delegator_agent**(cloud_names)	Invoke delegator_agent(cloud1,cloud2,cloud5)

2) Delegator Agent (DA): After finding the related clouds for the query, the delegator agents are created corresponding to the number of clouds to be searched. Again based on the number of services that are registered , each delegator can have one or more sub delegator agents. All sub delegator agents do their work in parallel mode. Each sub delegator agent performs matching operation and return the relevant services for the query. The working of delegator agent is given in TABLE III.

TABLE III
Delegator Agent

invoke **delegator_agent**(query,cloud_names)	invoke delegator_agent(R32,cloud1,cloud2,cloud5)
Create delegator_agents(cloud_names)	Create delegator_agents(cloud1,cloud2,cloud5) delegator_agent1
Create sub_delegator_agents(cloud_name)	Create subdelegator_agents(cloud1) sub_delegator_agents D11,D12
invoke matchmakingalg(query)	invoke matchmaking alg(volunteer(R32))
Retrieve relevant services	get_student_volunteers() get_nss_volunteers() get_schoolstudent_volunteers() etc

match(outA, outQ)

if outA = outQ **then**
return Exact

else if outQ superclass of outA **then**
return Plugin

else if outQ subsumes outA **then**
return Plugin

else if outA subsumes outQ **then**
return Subsumes

else
return Fail

Fig.4 Match Making

The improved match making algorithm is used by the matchmaking agent to find the match between services. The algorithm is shown in Fig.4.In that algorithm outA represents output for the registered (advertised) services. outQ represents the output for the query. The input, output, precondition and effect is also got form the user for each splitted query and it is matched with the registered services according to the matchmaking algorithm by using service ontology. Weight has been assigned to each class (Exact, Plugin, Subsume, Fail). The priority ranges from Exact to Fail. The cumulative weight for IO(input,output) and IOPE(input,output,pre condition, effect) is calculated and ranked by the IOPE based ranking agent. The service which has the highest score is more relevant to the query. The services which has the score greater than the threshold is sent to the QoS agent. Threshold value is calculated by taking the average of minimum rank and maximum rank.

3) Qos Agent: The pseudo code for the QoS agent is given in TABLE IV. The QoS agent collects all the relevant services from various delegator agents. It gets the availability value from the OWL-Q of each service and then the services are arranged in the descending order according to the availability value by the QoS based ranking agent. The retrieved services are displayed to the user.

TABLE IV
Qos Agent

invoke QoS_agent(delegator_agentname, list_of_services)	invoke QoS_agent(D1, list_of_services)
Get availabilty_values(list_of_services)	Get availabilty_values(list_of_services)
invoke ranking(list_of_services)	invoke ranking(list_of_services)
invoke display(services,cloud_name)	get_nss_volunteer() get_student_volunteer() get_ncc_volunteer() etc

4) Monitoring Agent: Monitoring Agent monitors semantic repository for the registration of new service from providers. If any new service is found by MA then it informs the Update Agent. The processing of monitor agent and update agent is given in TABLE IV.

5) Update Agent (UA): The Update Agent is responsible to modify the service history regarding the availability of services and also updates the semantic repository if new service is monitored by Monitoring Agent.

TABLE IV
Update Agent

For change in SLA/OWL-S/OWL-Q **Call Service MA ()** Monitor changes () Get Modify Credentials () Call Service UA () Reflect changes in Service Registry Reflect Changes in Cloud Ontology Registry Return to Monitoring Agent	**modify (Cloud1:get Biscuits ())** **OWL-Q (Cloud1: get Biscuits (has Availability))→ 0** **Call Service MA ()** Monitor changes () Get Modify Credentials () **Set(Cloud1: get Biscuits(has Availability) → 0** Call Service UA () Reflect changes in Service Registry Reflect Changes in Cloud Ontology Registry Return to Monitoring Agent

IV. PROTOTYPE IMPLEMENTATION

A. Experimental Setup

This section explains the tools and platforms involved in the implementation of the prototype. The Eucalyptus cloud has been setup to deploy all the services and the agent framework has been implemented to retrieve the services in the cloud environment.Net beans (Version 6.8) platform has been used for java based implementation of web services. Protégé (version 3.3.1) platform provides Protege OWL editor and OWLS editor. FUSION SDMX Registry (version 2.0) is used as a semantic registry. There are 500 services have been taken form OWLS TC Test collection. Fig.7 shows sequence of steps involved in ontology driven web service discovery in cloud. The following TABLE V describes the various domain related services registered under particular cloud. The cloud ontology is also shown in Fig.6.

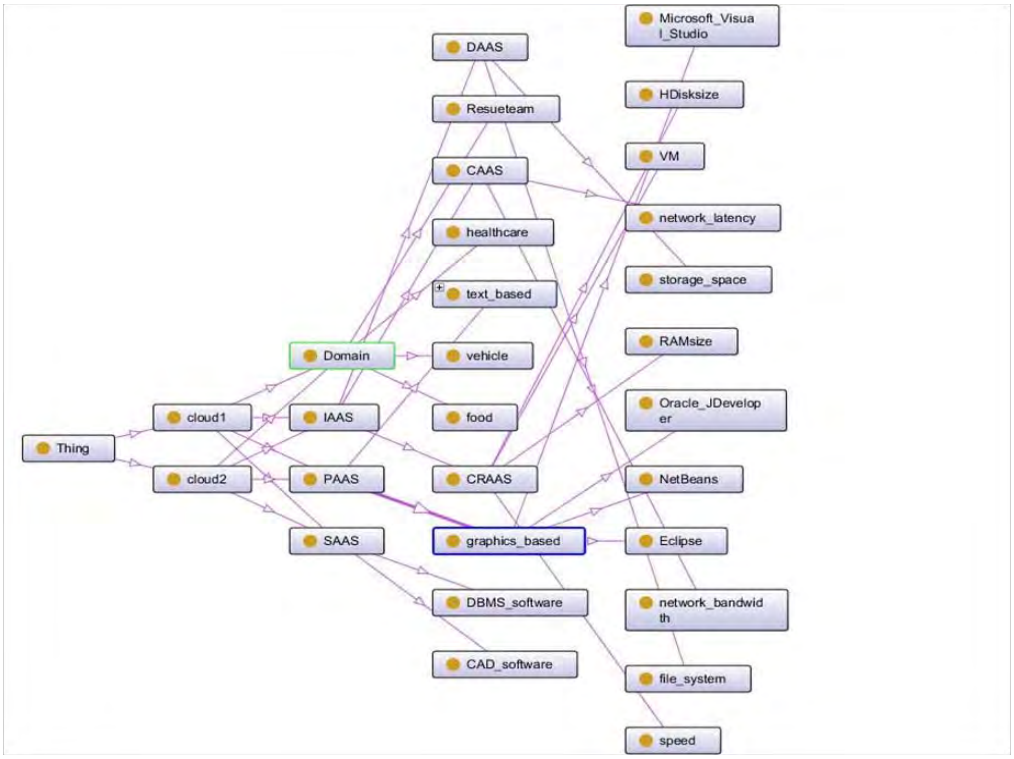

Fig.6. Cloud Ontology

TABLE V
Cloud with Domain of Services

Cloud	Domains
Cloud1	Food, Rescueteam
Cloud2	Food, Healthcare, rescueteam, Vehicle
Cloud3	Weather, food
Cloud4	Vehicle ,Healthcare
Cloud5	Rescueteam, Weather, Vehicle

B. Cloud Service Discovery

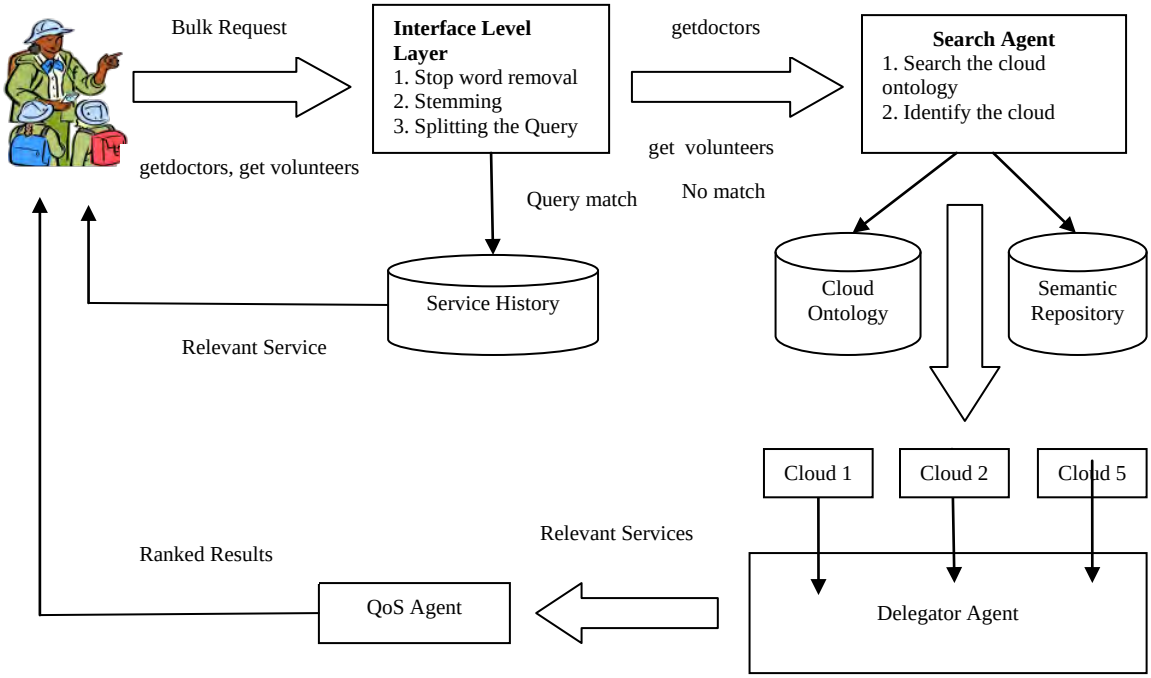

Fig. 7. Cloud Service Discovery

The user sends the bulk query that arises in emergency situation. In the above example, for an accident to treat the people there is a need for doctors and volunteers, the user query is got through the user interface and the query has been sent to interface lever layer agents. Then it does the operations on the bulk query like stemming, stop words removal and splitting of query. Therefore the query is splitted into two getvolunteer and getdoctor. The user query is matched against the services present in the service history. In this example there is no match between the requested query and stored services in the service history. Then the query is given to the Search Agent. The Search Agent searches the cloud ontology and finds the correct cloud to identify the services. The query getvolunteer is present in cloud1,cloud2 and cloud5. The relevant services that are present in these clouds should be found by using Delegator Agent. Delegator Agent gets the corresponding cloud from search agent and it find the relevant service in that cloud using IO and IOPE matching. The work of the delegator is parallelized using hadoop map reduce concept. There are three delegators for cloud2 because the no of services in cloud2 is large. For the example query getvolunteer the retrieval of services by several delegator agents are given in the TABLE VI. All the retrieved services that are relevant to the user query is given to the QoS agent. The coordinator agent ranks all the services based on their availability value and the ranked services are displayed to the user for approval of service execution.

TABLE VI
Delegator Agent

Cloud	Delegator	Total No of services for getvolunteer	No of retrieved services (IO)	No of retrieved services (IOPE)	Total No of services Retrieved (IO)	Total No of services Retrieved (IOPE)
Cloud1	D11	70	20	30	30	45
	D12		10	15		
Cloud2	D21	110	15	25	45	60
	D22		20	25		
	D23		10	10		
Cloud5	D31	80	20	30	50	65
	D32		30	35		

The service request and the sample results are shown in Fig 8. Even though the last service is the user requested service, but the delivered service is nssvolunteer() because the availability of the nccvolunteer() service is less when compared to nssvolunteer and the top two services have been chosen from cloud2. In an emergency situation if only the functional parameter like IO and IOPE is considered the relevant services can be given to the user. But the user doesn‘t know whether the services have been executed or not, if the non functional parameter like availability is also taken for consideration then there is a guarantee that only the Quality services are delivered to the user.

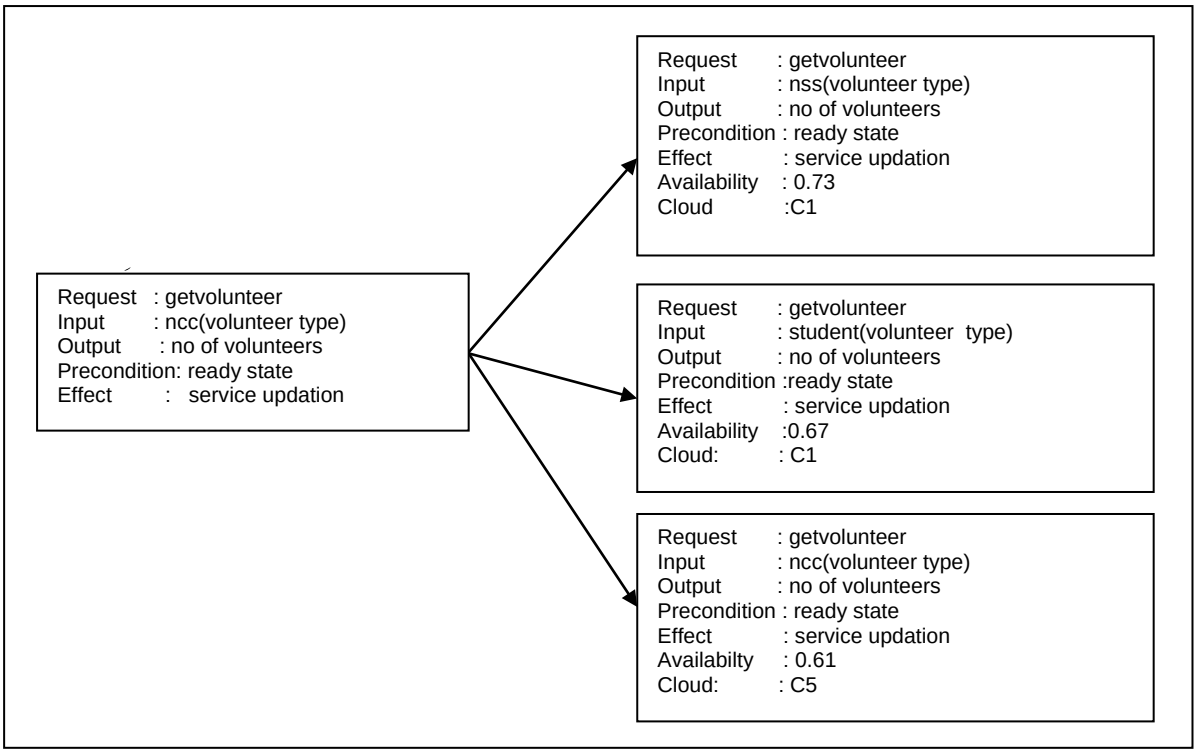

Fig. 8. Request and Results

C. Results

The following TABLE VII describes the results obtained for IO based matchmaking and IOPE based match making. These match making techniques are followed by the delegator agent to retrieve the services that are present in the cloud. After analysing the result (Fig.9) in IO method the recall is high but precision is low. In case of IOPE based service discovery the recall is low compared to IO but the precision is high than IO. It indirectly means that IOPE based service discovery results in delivering only less no of high relevant services to the user.

TABLE VII
IO Matching Vs IOPE Matching

Type of Matching	TP	FP	FN	TN	Recall	Precision	F-Measure
IO	80	40	20	140	0.8	0.66	0.723
IOPE	130	40	40	90	0.764	0.764	0.764

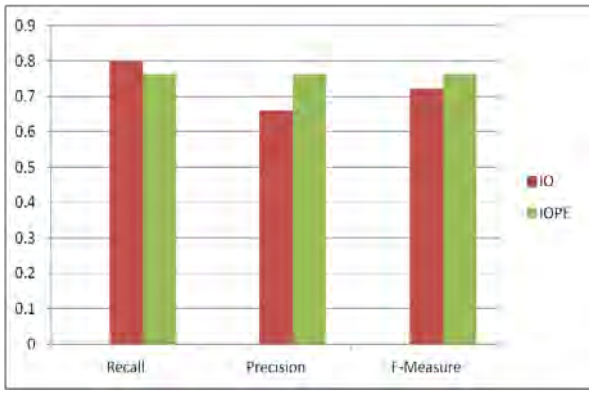

Fig.9. IO and IOPE results

V. CONCLUSION

As Cloud computing allows the users to create resources on-demand scales in response to the demands of victims of emergency situations, it is recommended as the preeminent solution. Availability is one of the major parameter to be considered during discovery. Since the cloud services are hosted at various geographical locations, the possibility of single point failure is avoided. The novelty and significance of this paper is that distributed and cooperative agents were used to create an ontology based self-organizing service discovery approach. Load balancing a significant influencing factor has been dealt using Delegation concept which allow for flexibility since they can expand quickly as the demands increases. This paper will help researchers to understand the need for integrating the technologies namely semantic web services, cloud computing. This in turn may lead to a synchronization of research efforts and more inter-operable Cloud technologies and services at the IaaS layer.

ACKNOWLEDGMENT

The authors express their gratitude to the AICTE for financial support under the Research Promotion Scheme.

REFERENCES

[1] Bhuvaneswari A,Dr.G.R.Karpagam, Ontology-Based Emergency Management System in a Social Cloud, International Journal on Cloud Computing: Services and Architecture(IJCCSA),Vol.1, No.3, November 2011.
[2] Anju Gautam and Rahul Pareek, Cloud Computing: An Overview, Journal of Global Research in Computer Science, Volume 2, No. 7,July 2011.
[3] Grigoris Antoniou,Frank van Harmelen, A Semantic Web Primer, Second Edition ,The MIT Press, Cambridge, 2003.
[4] Bettina Fazzinga,Thomas Lukasiewicz ,Semantic search on the Web, Semantic Web — Interoperability, Usability, Applicability, IOS Press, (2010) 1–7
[5] Ethan Cerami, Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL First Edition, O'Reilly February 2002.
[6] Taekgyeong Han and Kwang Mong Sim, "An Ontology-enhanced Cloud Service Discovery System", Proceedings of the International multi conference of Engineers and computer Scientists, 2010, Vol I, Hong Hong.
[7] Miranda Zhang, Rajiv Ranjan, Armin Haller,, Dimitrios Georgakopoulos,, Michael Menzel,, Surya Nepal, An Ontology-based System for Cloud Infrastructure Services' Discovery, 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing,2012 october.
[8] Jaeyong Kang, Kwang Mong Sim, Towards Agents and Ontology for Cloud Service Discovery, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011.
[9] G. Vadivelou, E. IIavarasan, R. Manoharan, P. Praveen, "A QoS Based Web Service Selection through Delegation, International Journal of Scientific & Engineering Research, Volume 2, Issue 5, May-2011.
[10] Umesh Bellur, Roshan Kulkarni, Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching, ICWS, pp-86-93(2007).
[11] Alireza Zohali, Dr.Kamran Zamanifar, Matching Model for Semantic Web Services Discovery, Journal of Theoretical and Applied Information Technology, 2005 - 2009 JATIT.
[12] Arkaitz Ruiz-Alvarez, Marty Humphrey, An Automated Approach to Cloud Storage Service Selection, ScienceCloud'11, 2011, San Jose, California, USA.

[13] Maheswari, S. and G.R. Karpagam, "QoS Based Efficient Web Service Selection". European Journal of Scientific Research, 66(3): p. 428-440, 2011.
[14] Noraida Haji Ali , Noor Syakirah Ibrahim, Porter Stemming Algorithm for Semantic Checking, ICCIT 2012.
[15] Katia Sycara,Massimo Paolucci,Julien Soudry,Naveen Srinivasan, Dynamic Discovery and Coordination of Agent-Based Semantic Web Services, IEEE Internet Computing, June 2004.
[16] Shangguang Wang, Zibin Zheng, Qibo Sun, Hua Zou and Fangchun Yang,"Cloud Model for Service Selection", IEEE INFOCOM 2011 Workshop on Cloud Computing.
[17] J. Octavio Gutierrez-Garcia and Kwang-Mong Sim," Self-Organizing Agents for Service Composition in Cloud Computing", 2nd IEEE Conference on Cloud Computing Technology and Science, 2010.
[18] C. Gershenson, and F. Heylighen, "When Can we Call a System Selforganizing?," in Proc. of the 7th European Conference,2003.
[19] G. Antoniou et al. Web Ontology Language: OWL. Handbook on Ontologies in Information Systems, 2003.
[20] D. Martin et al. OWL-S: Semantic Markup for Web Services. Technical Report,,W3Chttp://www.w3.org/Submission/2004/07/ ,2004.